WaterlooClarke: TREC 2015 Microblog Track

Mustafa Abualsaud, Milad Ghaznavi, Daniel Recoskie, and Charles L. A. Clarke
School of Computer Science, University of Waterloo, ON, Canada
{m2abuals,eghaznav,dprecosk,claclark}@uwaterloo.ca

I. Real Time Filtering Task

The goal of the TREC 2015 Microblog Track is to develop a real-time relevancy retrieval system that monitors a stream of social media posts and recommends relevant posts according to users' interests [1]. In this track, the representative social media is Twitter, and relevant posts are tweets with respect to a user's interest. A user's interest is represented by an *interest profile* containing a title, a description, and a narrative. Relevant tweets are recommended to the user in two tasks, *push notification* and *periodic email digest*.

A. Push Notification Task

The push notification task is meant to model a system that notifies the user in real-time on his/her mobile phone as it finds a relevant tweet. This notification, i.e. pushing a relevant tweet, is triggered in less than 100 minutes after the tweet creation time[1]. At most 10 tweets per day are pushed for each interest profile.

B. Periodic Email Digest Task

The periodic email digest task aggregates a ranked-list of up to 100 relevant tweets for each interest profile into an email and sends it to the user at the end of every day.

II. Tasks Evaluation

The developed systems listen to the Twitter sample stream[2] for a period of 10 days and report the relevant tweets. All reported tweets are evaluated by assessors on a four point scale: *redundant/spam/junk*, *not interesting*, *somewhat interesting*, and *very interesting*. Very interesting and somewhat interesting tweets have a gain of 1 and 0.5, respectively. The other two categories receive a gain of zero. Note that only English tweets are considered. Moreover, redundancy is determined by the clustering protocol from the tweet time-line generation (TTG) task 2014 [7]. This protocol groups tweets into semantic clusters. For a particular interest profile, only one tweet from each cluster gets credit. Tasks are evaluated as follows.

A. Evaluation of Push Notification Task

Pushed tweets for a particular day and interest profile are scored based on two temporally-discounted gain measures: *Expected Latency-discounted Gain (ELG)* [2] and *Normalized Cumulative Gain (NCG)*. ELG, defined in Eq. 1, is the primary measure. In Eq. 1, n is the number of pushed tweets for a particular day and interest profile. NCG is the secondary metric and defined in Eq. 2. The maximum possible gain is represented by Z. The gain term in both measures is determined by an assessor. The delay term, which temporally penalizes tweets, is measured in minutes and represents the difference between the tweet creation time and the push time. Note that if no interesting tweet exists for a day, pushing no tweets gets the maximum score of one. The score of a particular interest profile is the average of these daily scores across all days in the evaluation period. Finally, the score of the push notification task is the average of scores of all interest profiles.

$$\frac{1}{n}\sum_{i=1}^{n}\text{gain}(i)\cdot\max\left(0,100-\text{delay}\right) \tag{1}$$

$$\frac{1}{Z}\sum_{i=1}^{n}\text{gain}(i)\cdot\max\left(0,100-\text{delay}\right) \tag{2}$$

B. Evaluation of Periodic Email Digest Task

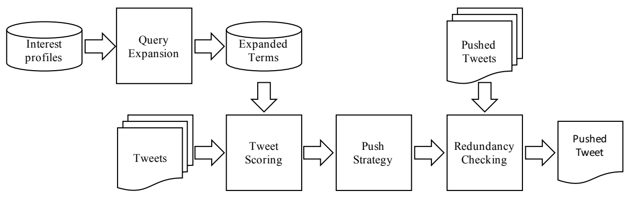
The periodic email digest of a particular day for a given interest profile is treated as a ranked list from which $nDCG@k$ [10] is computed. $nDCG@k$ is defined in Eq. 3. Note that $IDCG_k$ refers to the best possible DCG_k. The number of evaluated tweets k is determined by NIST. The score of an interest profile is the average of the $nDCG@k$ values across all evaluation days. The score of the email digest task is the average over all interest profiles.

$$nDCG@k=\frac{DCG_k}{IDCG_k}\quad\text{s.t.}\quad DCG_k=\sum_{i=1}^{k}\frac{2^{\text{gain}(i)}-1}{\log_2\left(i+1\right)} \tag{3}$$

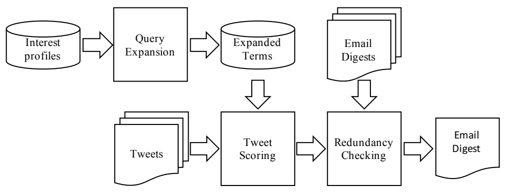
III. Developed Systems

Fig. 1 depicts the outline of our systems. As presented, the push notification (Fig. 1a) and the periodic email digest (Fig. 1b) systems share three major components: *Query Expansion*, *Tweet Scoring*, and *Redundancy Checking*. Both systems first expand the query terms of each interest profile. The expanded terms are then used to score the tweets. The push notification system has an additional push strategy component that decides whether and when a candidate tweet should be pushed. Finally, both systems avoid recommending redundant tweets by making use of a tweet similarity measure. Note that the push notification system recommends tweets in real time, whereas the periodic email digest system sends a daily email

[1]Time when the tweet was tweeted by the user.
[2]https://dev.twitter.com/streaming/public



(a) Push Notification System



(b) Periodic Email Digest System

Fig. 1: Developed Systems

digest. The individual components are described in more detail as follows.

A. Query Expansion

As mentioned earlier, each interest profile contains a title, description, and narrative. The titles are between one and eight words long. The descriptions are single sentences which briefly describe the topic. The narratives are paragraphs that contain more detail about the topic. Our system makes use of the title only, and discards the description and the narrative.

We begin by performing query expansion on the title using pseudo relevance feedback [8]. Stop words are removed and the remaining terms are passed to Twitter's search API[3]. Twitter's search API returns a list of tweets from the previous week where each tweet must contain the exact query terms. We treat the returned tweets as relevant to the interest profile. If we receive fewer than 100 tweets from the initial search, we make use of the Whoosh[4] library to generate syntactic variations of the original title in order to perform additional searches. We then calculate term frequencies, q_i, based on the returned tweets as follows:

$$q_i = \frac{n_i}{n} \tag{4}$$

where n_i is the number of tweets containing term i and n is the total number of tweets. We discard stop words and any term that occurs in less than three tweets. We then calculate the prior frequencies, p_i, of each term in the same manner as q_i, but using a corpus of previously collected tweets. Our corpus was collected over approximately seven days from the Twitter sample stream. We then score each term by the following formula.

$$score(i) = q_i \log \frac{q_i}{p_i} \tag{5}$$

The top five terms are chosen as expanded terms. In the case where the Twitter search API does not return an adequate number of tweets, we make use of the Google Custom Search API[5]. We use the interest profile title as a query and consider the top 50 results as relevant. We use the snippets of each returned result as our documents, and rank terms in the same manner as above.

[3]https://dev.twitter.com/rest/public/search
[4]https://bitbucket.org/mchaput/whoosh/wiki/Home
[5]https://developers.google.com/custom-search/

B. Tweet Scoring

In order to rank tweets we developed two scoring functions which determine tweet relevancy. Consider interest profile p and tweet t. The scoring functions assign a score to the tweet as follows:

$$s(p,t) = \sum_{k \in T_p \cap T_t} w(k,p) \tag{6}$$

where T_p is the set of terms in the title together with the expanded terms from profile p, and T_t is the set of terms in tweet t. The function w assigns a weight to each term. Our first scoring function, s_{equal}, sets $w(k,p) = 1$ if the term k is in either the title or expanded terms of profile p and is zero otherwise. Our second scoring function, $s_{weighted}$, sets $w(k,p) = 1$ if term k is in the expanded terms of profile p. If term k is in the title of profile p, then $w(k,p) = \frac{15}{n}$ where n is the number of terms in the title.

C. Redundancy Checking

To avoid redundancy, we define the *Normalized Similarity Measure (NSM)* for two tweets. NSM is a real number in $[0,1]$. A value of zero means no similarity, whereas a value of one means equivalence. The following procedure computes the NSM of two tweets t_1, t_2:

1) Extract URLs in t_1, t_2
2) If there is an identical URL, return 1.0
3) Remove URLs and Twitter usernames from t_1, t_2
4) Tokenize and stem t_1, t_2 and store the terms in T_1, T_2
5) Remove stemmed stop words from T_1, T_2
6) Return $\frac{\text{Number of identical terms in } T_1, T_2}{\max(\text{Length of } T_1, \text{Length of } T_2)}$

Before pushing for each interest profile, we measure similarity of the candidate tweet to the already pushed tweets using NSM. If any of measured NSMs is higher than the threshold of 0.45, we assume that the candidate tweet is redundant, and we do not push it. We used the value of 0.45 based on empirical studies on candidate tweets suggested by our system.

D. Push Strategy

In the case of the push notification task, our system must not only decide what tweets are relevant enough to push, but also determine when to push them. Since there is a time delay penalty for pushing tweets, we must be able to identify high quality tweets shortly after we receive them from the stream. We address this issue by considering the secretary problem [6] and the hiring problem [9]. The secretary problem is defined as follows: we receive a stream of job applicants and interview them in the order of appearance. The goal is to to hire the best applicant (or best k applicants), but we have a constraint that we must hire an individual directly after we interview them. In the hiring problem there is no limit to the number of applicants hired, and the goal is to balance the trade-off between rate of hiring and quality of hired applicants. A push strategy has to consider challenges of both the secretary problem and hiring problem, namely selecting the top $k \leq 10$ applicants with highest qualities and hiring in a reasonable rate with the minimum delay penalty.

These problems closely model the scenario of the push notification task. As such, we make use of the following strategy for the secretary problem [3]:

1) Observe and rank the first $\lfloor \frac{n}{e} \rfloor$ applicants.
2) Set T to be the top k applicants, sorted by rank.
3) When a new applicant is observed and they are ranked higher than the lowest applicant in T, hire the applicant and remove the lowest applicant in T.
4) Repeat step 3 until k applicants are hired or there are no more applicants.

This algorithm is easily adapted to our task. In our case $k = 10$, that is, we wish to push up to ten tweets per interest profile. We then push any tweet if it scores higher than the lowest ranking tweet in T. This means our strategy will not incur any time delay penalties since we push tweets immediately after scoring them. Note that the algorithm will not consider any tweets in the first $\lfloor \frac{n}{e} \rfloor$ fraction of tweets. This is undesirable because we may miss out on high quality tweets that occur within this window. To solve this problem we modify step one of the algorithm. Instead of observing the first $\lfloor \frac{n}{e} \rfloor$ tweets each day, we determine T from the top k tweets from the previous day.

Lastly, we only push tweets if they are over a threshold score. For s_{equal} we set the threshold to three. For $s_{weighted}$ we set the threshold to $\frac{15}{n}$ where n is the number of terms in the profile title. These thresholds were determined empirically by evaluating our system on a set of test interest profiles that we created.

IV. Runs

We submitted two runs for both the push notification and periodic email digest tasks. We describe them below.

A. Push Notifications

Both runs make use of the algorithm described earlier. However, they each differ in the scoring function that is employed.

- UWCMBP1 makes use of s_{equal}
- UWCMBP2 makes use of $s_{weighted}$

B. Periodic Email Digest

Each run of the periodic email digest task returns the top 100 tweets per day as determined by the scoring functions mentioned earlier. Similarly to the push notification runs,

- UWCMBE1 makes use of s_{equal}
- UWCMBE2 makes use of $s_{weighted}$

V. Future Works

There are opportunities in improving the scoring functions, extending the redundancy checker, and exploring different push strategies. Our scoring functions make use of the text of each tweet. However, there are other pieces of data in each tweet, such as user data, hash-tags, embedded URLs, and Geo-locations. The scoring functions can probably be improved by incorporating some of this meta-data. Furthermore, our systems employs score thresholds that were determined empirically. These thresholds could be estimated more accurately by examining a larger collection of tweets over a longer period of time.

We believe that our push strategy is a novel application of the secretary and hiring problems. There are other more complex strategies considered in the literature [4], [5]. Applying these strategies to the push notification task is an interesting avenue for future research.

Lastly, our system treats each interest profile equally. In future, the scoring functions and push strategies can be customized per interest profile basis. For example, if we encounter an interest profile that has many relevant tweets per day, a higher score threshold can be set to be more conservative and recommend top tweets.

References

[1] Real-time filtering task guidelines. https://github.com/lintool/twitter-tools/wiki/TREC-2015-Track-Guidelines.

[2] Javed Aslam and Tetsuya Sakai. Trec 2013 temporal summarization.

[3] Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. Online auctions and generalized secretary problems. *ACM SIGecom Exchanges*, 7(2):7, 2008.

[4] F Thomas Bruss and Guy Louchard. Finding the κ best out of n rankable objects. a consecutive thresholds algorithm. 2013.

[5] Michael Dinitz. Recent advances on the matroid secretary problem. *SIGACT News*, 44(2):126–142, June 2013.

[6] PR Freeman. The secretary problem and its extensions: A review. *International Statistical Review/Revue Internationale de Statistique*, pages 189–206, 1983.

[7] Jimmy Lin, Miles Efron, Yulu Wang, and Garrick Sherman. Overview of the trec-2013 microblog track (notebook draft).

[8] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[9] RJ Vanderbei. The optimal choice of a subset of a population. *Mathematics of Operations Research*, 5(4):481–486, 1980.

[10] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, and Wei Chen. A theoretical analysis of ndcg type ranking measures. *arXiv preprint arXiv:1304.6480*, 2013.